

УДК 004.4

ОПТИМИЗАЦИЯ ХВОСТОВОЙ РЕКУРСИИ ФПЯП «ПИФАГОР»**Васильев В.С.,****научный руководитель д-р техн. наук Легалов А.И.*****Сибирский федеральный университет***

Одним из широко распространённых методов оптимизации программ является замена хвостовой рекурсии циклом [1, 2, 3]. Планируется реализация такого метода оптимизации для языка программирования «ПИФАГОР»[4], основной идеей которого является написание архитектурно-независимых программ. Замена рекурсии циклом позволит применять к ПИФАГОР-программам широкий класс оптимизирующих преобразований циклов [5].

Рекурсия функции является хвостовой, если после рекурсивного вызова в вызывающей функции не производятся вычисления. В большинстве языков, при хвостовой рекурсии за каждым рекурсивным вызовом стоит оператор, завершающий выполнение функции [1].

В языке ПИФАГОР ветвление реализуется посредством задержанных списков, поэтому даже в функциях, обладающих хвостовой рекурсией на путях между рекурсивным вызовом и оператором *return* могут встречаться операторы интерпретации, группировки в список и т. п.

ПИФАГОР является потоковым языком, поэтому определить является ли функция рекурсивной возможно лишь при помощи динамического анализа (выполнения программы), так например, синтаксически операция взятия элемента списка не отличается от операции вызова функции.

Функция вычисления факториала *fact* является леворекурсивной и не может быть оптимизирована описываемым методом.

// fact - факториал без хвостовой рекурсии

```
fact << funcdef X {
  [((X, 0) : [<,>=]) : ?] ^ (
    {"err: fact arg < 0"},
    {
      [((X, 1) : [<=, >]) : ?] ^ (
        {1},
        {
          (X, (X, 1) : - : fact) : *
        }
      ):.
    }
  ):. >> return
}
```

Функция *factTail* также возвращает факториал числа, однако, является праворекурсивной.

// factTail - факториал с хвостовой рекурсией

// X : 1 - аргумент

// X : 2 - промежуточный результат

```
factTail << funcdef X {
  INum << X : 1;
  TR << X : 2;
  [((INum, 0) : [<,>=]) : ?] ^ (
    { "err: fact arg < 0" },
    {
      [((INum, 1) : [<=, >]) : ?] ^ (
        { TR },
        {
          ((INum, 1) : -, (TR, INum) : *) : factTail
        }
      ) : .
    }
  ) : . >> return
}
```

В функции *factTail* результат рекурсивного вызова помещается сначала в задержанный список, затем в список данных, к которому применяется результат операции «?». Операция «?» возвращает целое число, в результате применения которого в списке данных выбирается соответствующий элемент — в данном случае, элементом является задержанный список.

Таким образом, в функции *factTail*, как и в функции *fact* результаты рекурсивного вызова группируются в списки, которые каким-то образом обрабатываются, значит, программно определить тип рекурсии функции крайне трудно.

Для замены хвостовой рекурсии в функции $f(X)$ циклом предлагается заменять операцию интерпретации аргумента Y функцией f , новой операцией *cycle*, которая должна

- очистить состояние функции f ;
- инициализировать аргумент X значением Y ;
- передать управление в начало функции f .

Очистка состояния подразумевает уничтожение всех вычисленных значений слоя данных функции, установка всех автоматов функции в начальное состояние.

Рассмотрим оптимизацию хвостовой рекурсии на примере функций реверса списка (*rev* и *revTail*).

В функции *rev* результаты рекурсивного вызова обрабатываются — к ним применяется операция преобразования списка данных в параллельный список, поэтому рекурсия функции *rev* не является хвостовой.

```
// rev - реверс списка без хвостовой рекурсии
// X : 1 - исходный список
// X : 2 - количество обработанных элементов
rev << funcdef X {
  IList << X : 1;
  Counter << X : 2;
  Len << IList : |;

  [((Counter, Len) : [>=, <]) : ?] ^ (
```

```

        {()},
        {
            (IList : Counter, (IList, (Counter, 1) : +) : rev : [])
        }
    ):. >> return
}

```

В функция *revTail* результаты рекурсивного вызова обрабатываются лишь оператором *return*, поэтому рекурсия является хвостовой и может быть оптимизирована.

```

// revTail - реверс списка с хвостовой рекурсией
// X : 1 - исходный список
// X : 2 - количество обработанных элементов
// X : 3 - промежуточный результат
revTail << funcdef X {
    IList << X : 1;
    Counter << X : 2;
    TOList << X : 3;
    Len << IList : |;

    [((Counter, Len) : [>=, <]) : ?] ^ (
        { TOList },
        {
            (IList, (Counter, 1) : +, (TOList : [], IList : Counter)) : revTail
        }
    ):. >> return
}

```

Для вызова *revTail(((4,5,6), 0, ()))* последовательность действий будет примерно следующей:

1. $X = ((4,5,6), 0, ());$
2. вычисляется $IList \leftarrow (4,5,6); Counter \leftarrow 0; TOList \leftarrow (); Len \leftarrow 3$
3. $Counter < Len$, выполняется cycle $((4,5,6), 1, (4))$, управление передаётся в начало *revTail*, аргументу *X* присваивается значение $((4,5,6), 1, (4))$;
4. вычисляется $IList \leftarrow (4,5,6); Counter \leftarrow 1; TOList \leftarrow (4); Len \leftarrow 3$;
5. $Counter < Len$, выполняется cycle $((4,5,6), 2, (5,4))$, управление передаётся в начало *revTail*, аргументу *X* присваивается значение $((4,5,6), 2, (5, 4))$;
6. вычисляется $IList \leftarrow (4,5,6); Counter \leftarrow 1; TOList \leftarrow (5,4); Len \leftarrow 3$;
7. $Counter < Len$, выполняется cycle $((4,5,6), 3, (6,5,4))$, управление передаётся в начало *revTail*, аргументу *X* присваивается значение $((4,5,6), 3, (6,5,4))$;
8. вычисляется $IList \leftarrow (4,5,6); Counter \leftarrow 3; TOList \leftarrow (6,5,4); Len \leftarrow 3$;
9. $Counter \geq Len$, выполняется $\{(6,5,4)\} :. \gg return$.

Оптимизация выполняется не над исходным кодом функции, а над ее промежуточным представлением. На *рис. 1*, *рис. 2* показаны фрагменты информационных графов, соответствующие функциям *rev* и *revTail* до оптимизации, на *рис. 3* показан фрагмент оптимизированного графа *revTail*.

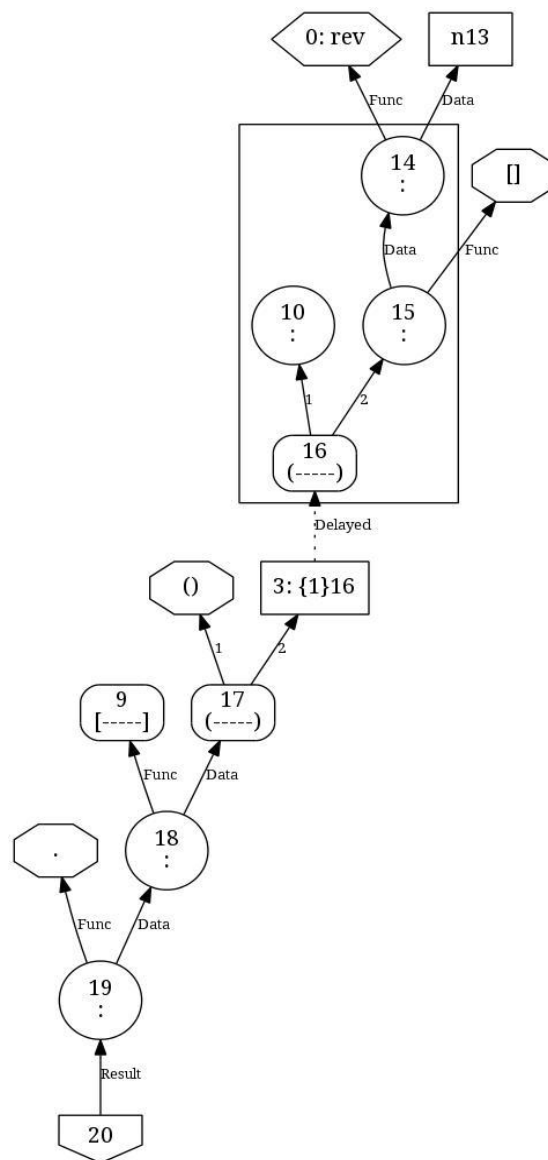


рис. 1

фрагмент ИГ функции rev

На *рис.1* видно, что в узле 15 выполняется интерпретация результатов рекурсивного вызова (узел 14), поэтому рекурсия не является хвостовой и не поддаётся оптимизации.

Список использованной литературы

1. Tail Recursion <http://c2.com/cgi-bin/wiki?TailRecursion>
2. Tail Call Optimisation in Common Lisp Implementations
<http://0branch.com/notes/tco-cl.html>
3. Tail recursion http://www.haskell.org/haskellwiki/Tail_recursion
4. Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии : журнал. — 2005. — Т. 10. — № 1. — С. 71-89.
5. Бутов А. Э. Автоматическое разрезание и слияние программных циклов // Магистерская диссертация. — Ростов-на-Дону, 2005 г.